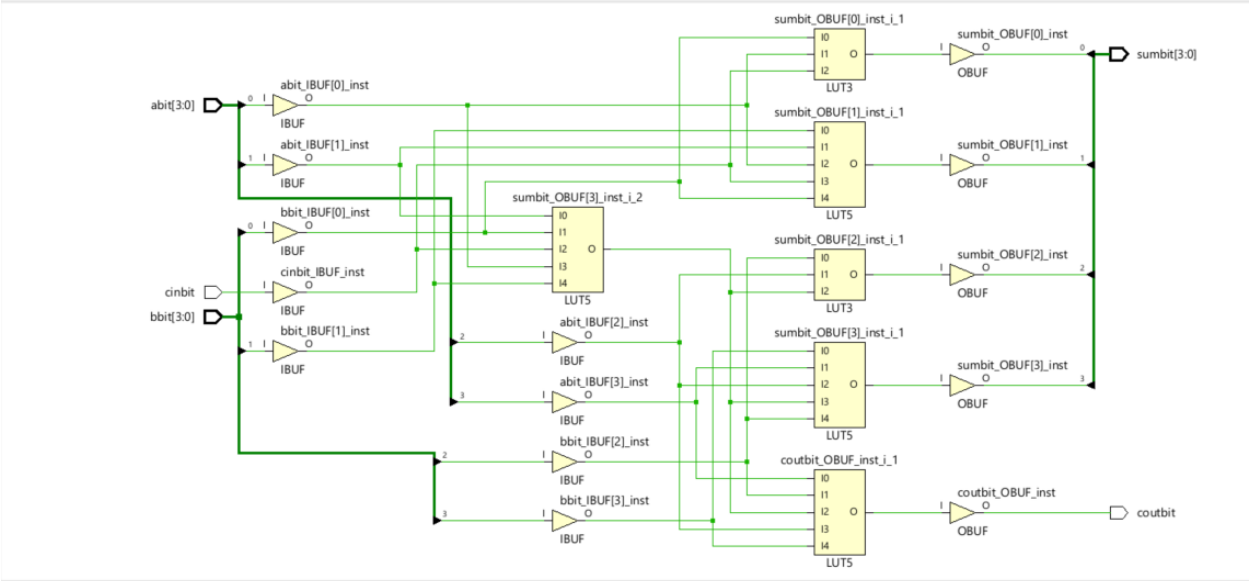
ECE3300L Lab2 Group H Report (Sherwin Sathish & Mohamed Hamida)

SCHEMATIC(4-bitFA):



This schematic illustrates a 4-bit full adder. There are buffers for the inputs and outputs. There are 14 input and output ports that were addressed in the constraint file (with the inputs being switches and the outputs being LEDs) as shown below:

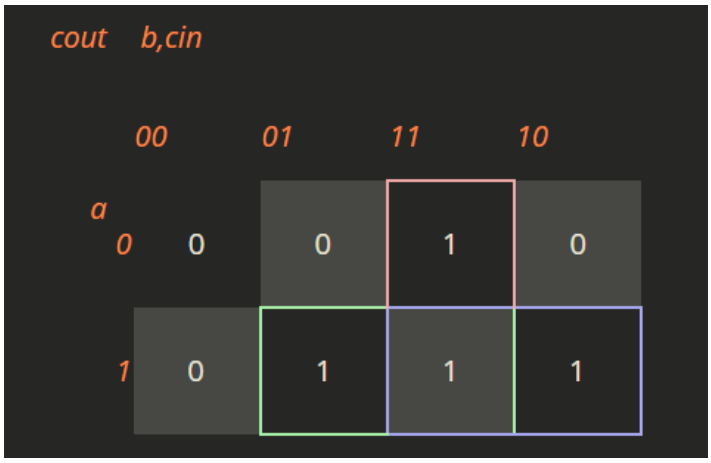
Xdc for 4-bit FA:

##Switches
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { abit[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { abit[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { abit[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15 IOSTANDARD LVCMOS33 } [get_ports { abit[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17 IOSTANDARD LVCMOS33 } [get_ports { bbit[0] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18 IOSTANDARD LVCMOS33 } [get_ports { bbit[1] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18 IOSTANDARD LVCMOS33 } [get_ports { bbit[2] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13 IOSTANDARD LVCMOS33 } [get_ports { bbit[3] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8 IOSTANDARD LVCMOS18 } [get_ports { cinbit }]; #IO_L24N_T3_34 Sch=sw[8]
LEDs
set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { sumbit[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { sumbit[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]

```
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { sumbit[2] }];
#IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { sumbit[3] }]; #IO_L8P_T1_D11_14
Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { coutbit }]; #IO_L7P_T1_D09_14
Sch=led[4]
```

Code Detail:

```
`timescale 1ns / 1ps
module MAJORITY(
    input x,
    input y,
    input z,
    output p
    );
  assign p = (y&z)|(x&z)|(x&y);
endmodule
```



In this module x represents a, y represents b, z represents cin, and p represents cout. Since cout is the output in the majority encoder we have to create a kmap that can help implement the necessary gates and assign a value to p. The majority encoder applies to cout because the cout depends on the majority of the binary input values.

```
`timescale 1ns / 1ps
module MAJORITY_TB(

    );
    reg x_tb,y_tb,z_tb;
    wire p_tb;
    MAJORITY COMP
    (
    .x(x_tb),
    .y(y_tb),
    .z(z_tb),
```

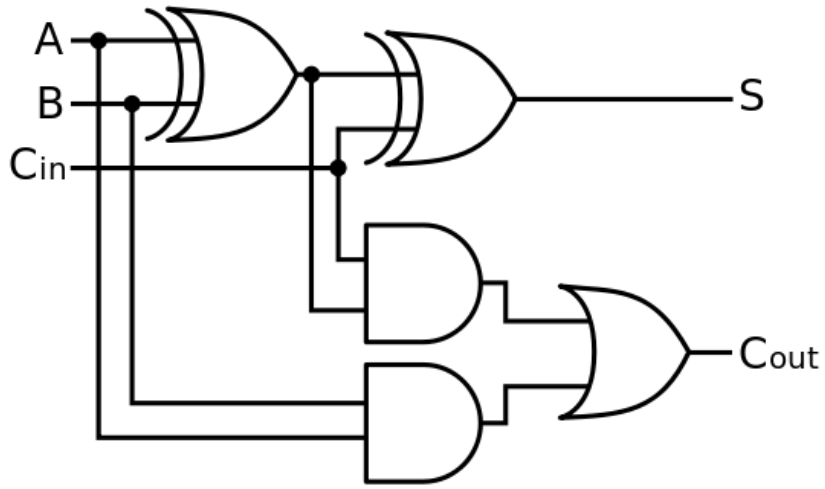
```
    .p(p_tb)
    );
     initial
          begin:  TST1
            x_tb = 1'b0;
            y_tb = 1'b0;
            z_tb = 1'b0;
           #10
            x_tb = 1'b0;
            y_tb = 1'b1;
            z_tb = 1'b1;
           #10
            x_tb = 1'b1;
            y_tb = 1'b0;
            z_tb = 1'b0;
           #10
            x_tb = 1'b1;
            y_tb = 1'b0;
            z_tb = 1'b1;
          #10
            x_tb = 1'b1;
            y_tb = 1'b1;
            z_tb = 1'b1;
          #1000
          $finish;
          end
endmodule
```

Here for the testbench we instantiated the values for the testbench by calling the values for the MAJORITY class, and tested different inputs to verify that the majority encoder works.

```
`timescale 1ns / 1ps
module FA(
    input a,
    input b,
    input cin,
    output sum,
    output cout
    );
    wire temp;
    xor(temp,a,b);
    xor(sum,temp,cin);

    MAJORITY UNIT
    (
    .x(a),
    .y(b),
    .z(cin),
    .p(cout)
    );

endmodule
```



Here for the FA module we call the MAJORITY class and we use gate level implementation to add the 2 xor gates to yield the sum, which completes the full adder. The temp wire is the output of the xor gate with inputs a and b. This temp wire and cin are used to form the xor gate for the sum. This creates the 1-bit FA product.

```
`timescale 1ns / 1ps
module FA_TB(

    );
    reg a_tb,b_tb,cin_tb;
    wire sum_tb,cout_tb;

    FA COMP
    (
    .a(a_tb),
    .b(b_tb),
    .cin(cin_tb),
    .sum(sum_tb),
    .cout(cout_tb)
    );
     initial
          begin:  TST1
            a_tb = 1'b1;
            b_tb = 1'b0;
            cin_tb = 1'b0;
           #10
            a_tb = 1'b0;
            b_tb = 1'b1;
            cin_tb = 1'b1;
           #10
            a_tb = 1'b1;
            b_tb = 1'b0;
            cin_tb = 1'b0;
           #10
            a_tb = 1'b1;
            b_tb = 1'b0;
```

```
        cin_tb = 1'b1;
      #10
        a_tb = 1'b1;
        b_tb = 1'b1;
        cin_tb = 1'b1;
      #1000
      $finish;
      end

endmodule
```

Here for the testbench we instantiated the values for the testbench by calling the values for the FA class, and tested different inputs to verify that the 1-bit FA works.

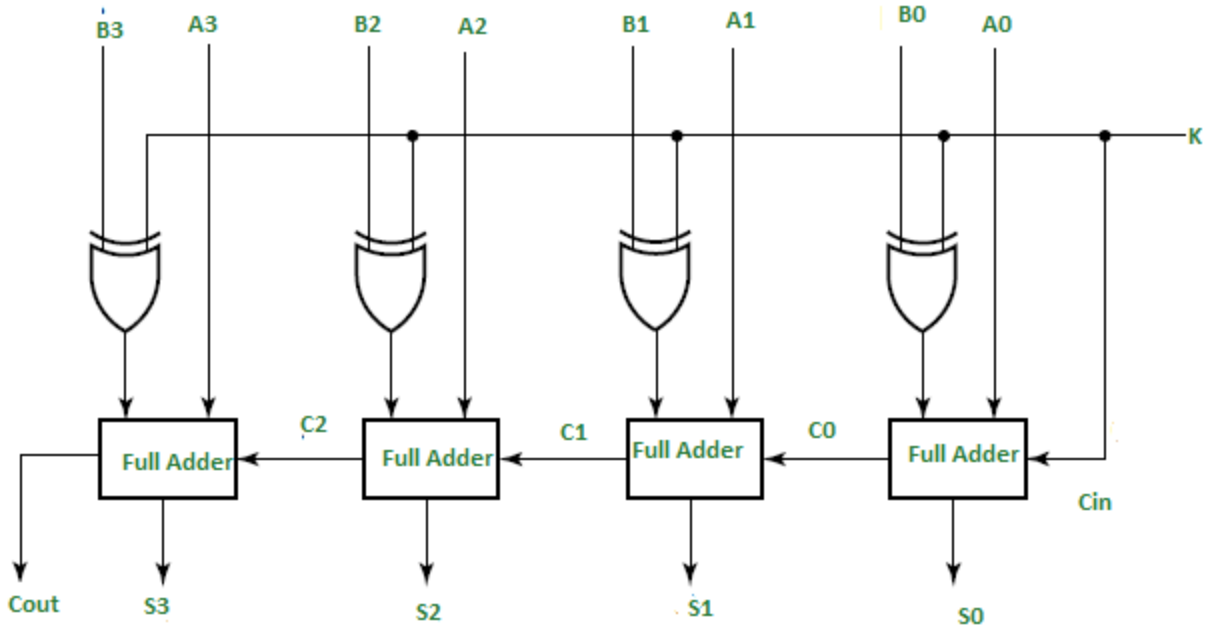
```
`timescale 1ns / 1ps
module FAnbit
      #(parameter DT = 8)
(
   input cinbit,
   input [DT-1:0] abit,
   input [DT-1:0] bbit,
   output [DT-1:0] sumbit,
   output coutbit
   );
   wire [DT-1:0] bridge;

     FA UNIT0 (
            .cin(cinbit),
            .a(abit[0]),
            .b(bbit[0]),
            .sum(sumbit[0]),
            .cout(bridge[0])
              );

   genvar i;
   generate
   for(i=1;i<DT;i=i+1)
      begin

      FA UNIT(
            .cin(bridge[i-1]),
            .a(abit[i]),
            .b(bbit[i]),
            .sum(sumbit[i]),
            .cout(bridge[i])
      );
    end
   endgenerate

   assign coutbit=bridge[DT-1];
endmodule
```



The FAnbits makes the full adder flexible for any amount of inputs. We instantiated the FA for the initial conditions in order to make it flexible for later. The for loop is for calling the FA module to create however many full adders we need. The bridge wire was created to establish the connection between each full adders' cin and cout. The assign at the end takes care of the final cout wire. The parameter DT allows for flexibility because we can choose how many bit full adders we want.

`timescale 1ns / 1ps

```
module FAnbit_tb
#(parameter DT_TB=4)
(

   );

 reg [DT_TB-1:0] abit_tb, bbit_tb;
 reg cinbit_tb;


 wire [DT_TB-1:0] sumbit_tb;
 wire coutbit_tb;

 FAnbit
 #(
  .DT(DT_TB)
 )

  FA_TB_NBITS
  (
```

```
            .cinbit(cinbit_tb),
            .abit(abit_tb),
            .bbit(bbit_tb),
            .sumbit(sumbit_tb),
            .coutbit(coutbit_tb)
    );

    initial
        begin:  TST1
          abit_tb = 1;
          bbit_tb = 3;
          cinbit_tb = 0;
          #10
          abit_tb = 4;
          bbit_tb = 6;
          cinbit_tb = 0;
         #10
          abit_tb = 8;
          bbit_tb = 8;
          cinbit_tb = 0;


        #1000
        $finish;
        end
endmodule
```

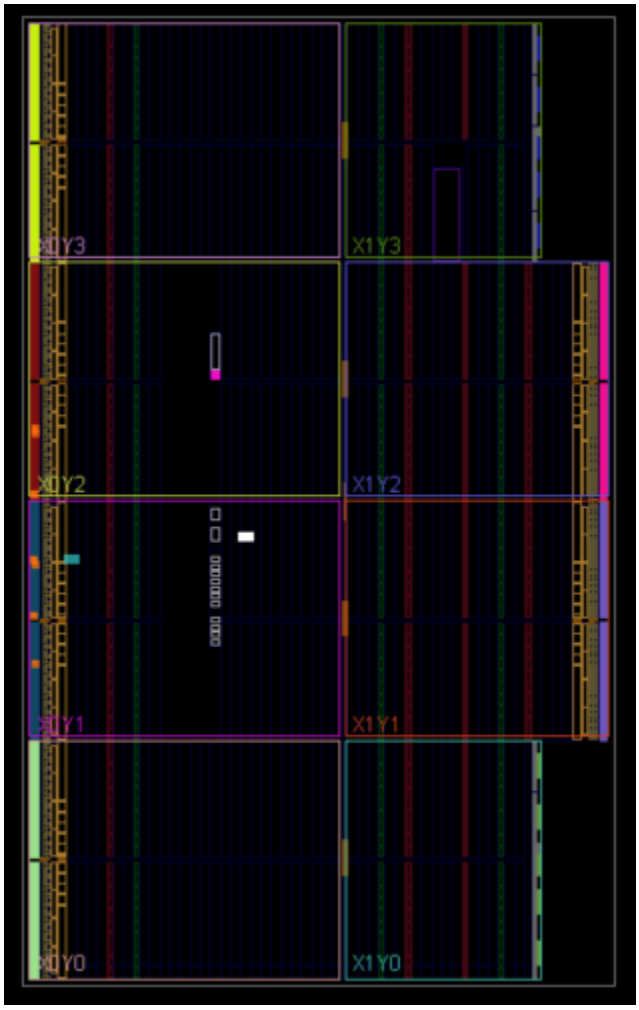
Here for the testbench we instantiated the values for the testbench by calling the values for the FAnits class, and tested different inputs to verify that the n-bit FA works.

For all the testbenches the inputs were reg(store values) and the output was wire.

Corner Cases/Error :

One corner case can apply for the n-bit FA testbench if you put an input b of 20 which requires more bits than available, an input of 4 will be shown because the MSB got cut off. However, in order to fix this the only thing that needs to be done is increase the parameter size of the nbits so that 20 will fit in the bit size, and will actually input the correct value of 20.

IMPLEMENTED DESIGN:



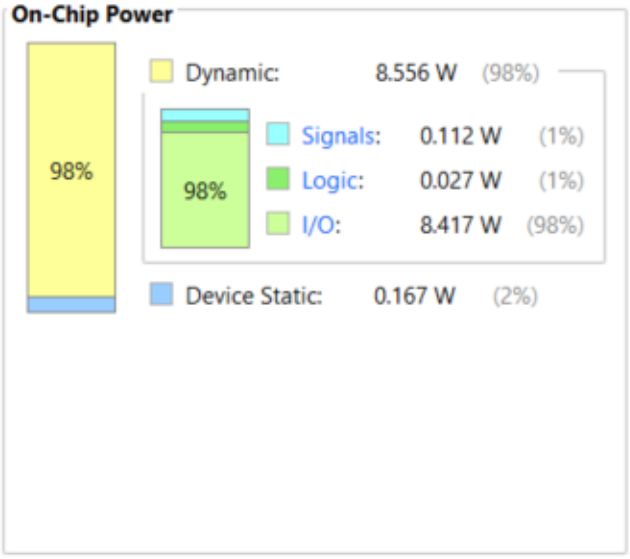
POWER SUMMARY:

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**8.723 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**64.8°C**
Thermal Margin:	20.2°C (4.4 W)
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



UTILIZATION:

Name	Slice LUTs (63400)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)
N FAnbit	4	1	4	14

RESOURCE USAGE:

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP
✓ synth_1	constrs_1	synth_design Complete!									4	0	0	0	0
✓ impl_1	constrs_1	write_bitstream Complete!	NA	NA	NA	NA		NA	8.723	0	4	0	0	0	0